МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

КУРСОВАЯ РАБОТА

по дисциплине «Алгоритмы и Структуры Данных»

Тема: Сортировки. Демонстрация

Студент гр. 8304 _____ Нам Ё Себ

Преподаватель _____ Фирсов М.А.

Санкт-Петербург

2019

ЗАДАНИЕ

НА КУРСОВУЮ РАБОТУ

Студент Нам Ё Себ

Группа 8304

Тема работы: сортировки

Исходные данные:

Написать программу для реализации сортировки расческой.

Содержание пояснительной записки:

- Содержание
- Введение
- Сортировка расческой
- Тестирование
- Исходный код
- Использованные источники

Дата выдачи задания: 11.10.2019

Дата сдачи реферата:

Дата защиты реферата: 25.12.2019

Студент _____ Нам Ё Себ

Преподаватель _____ Фирсов М.А.

АННОТАЦИЯ

В данной работе была создана программа на языке программирования C++, которая сочетает в себе несколько функций: ввода/вывода массива и его сортировки. Были использованы преимущества C++ для минимизации кода. Для лучшего понимания кода было в нем приведено большое кол-во комментариев и отладочных выводов. Также была проведена его оптимизация с целью экономии выделяемой в процессе работы памяти и улучшения быстродействия программы.

SUMMARY

In this work, a program was created in the C ++ programming language, which combines several functions: input / output of an array and its sorting. The benefits of C ++ were used to minimize code. For a better understanding of the code, a large number of comments and debugging conclusions were given in it. Also, its optimization was carried out in order to save the memory allocated in the process of working and improve the speed of the program.

СОДЕРЖАНИЕ

Введение

Целью данной курсовой работы является реализация сортировки расческой. Сортировки применяются во многих областях, поэтому создание эффективной сортировки является приоритетной целью. Существует много видов сортировок, однако лишь малая часть из них применяется в виду худших показателей скорости работы.

1. СОРТИРОВКА РАСЧЕСКОЙ

Сортировка расчёской улучшает сортировку пузырьком, и конкурирует с алгоритмами, подобными быстрой сортировке. Основная идея — устранить *черепах*, или маленькие значения в конце списка, которые крайне замедляют сортировку пузырьком (*кролики*, большие значения в начале списка, не представляют проблемы для сортировки пузырьком).

В сортировке пузырьком, когда сравниваются два элемента, промежуток (расстояние друг от друга) равен 1. Основная идея сортировки расчёской в том, что этот промежуток может быть гораздо больше, чем единица (сортировка Шелла также основана на этой идее, но она является модификацией сортировки вставками, а не сортировки пузырьком).

Основная идея «расчёски» в том, чтобы первоначально брать достаточно большое расстояние между сравниваемыми элементами и по мере упорядочивания массива сужать это расстояние вплоть до минимального. Таким образом, мы как бы причёсываем массив, постепенно разглаживая на всё более аккуратные пряди. Первоначальный разрыв между сравниваемыми элементами лучше брать с учётом специальной величины, называемой фактором уменьшения, оптимальное значение которой равно примерно 1,247. Сначала расстояние между элементами равно размеру массива, разделённого на фактор уменьшения (результат округляется до ближайшего целого). Затем, пройдя массив с этим шагом, необходимо поделить шаг на фактор уменьшения и пройти по списку вновь. Так продолжается до тех пор, пока разность индексов не достигнет единицы. В этом случае массив досортировывается обычным пузырьком.

Худшее время: O(n^2)

Среднее время: $\Omega(n^2/2^p)$

Лучшее время: O(nlogn)

Затраты памяти: O(1)

1. ФУНКЦИИ И СТРУКТУРЫ ДАННЫХ

Для считывания данных из входного файла была реализована функция readFileData, которая при помощи конструкции while (std::getline(in, currentFileString)), осуществляет запись строк из входного потока in в строчный массив, который был ранее принят по ссылке.

Для определения типа данных хранящихся в массиве был реализована функция determineType, которая осуществляет последовательное преобразование входной строки в другие типы. В случае успешного определения типа функция возвращает его код, в противном случае тип данных остается строчным.

Далее в зависимости от ранее определенного типа осуществляется формирование массива, которое осуществляется при помощи функции split, она, в свою очередь, при помощи последовательного перебора эл-ов массива во входной строке формирует массив.

Далее осуществляется сортировка массива.

2. ИНТЕРФЕЙС ПРОГРАММЫ

1. Ввод – файловый, пути до файлов передаются как аргументы командной.
2. Отладочные выводы, полученные в ходе выполнения, представлены на рисунке 2.

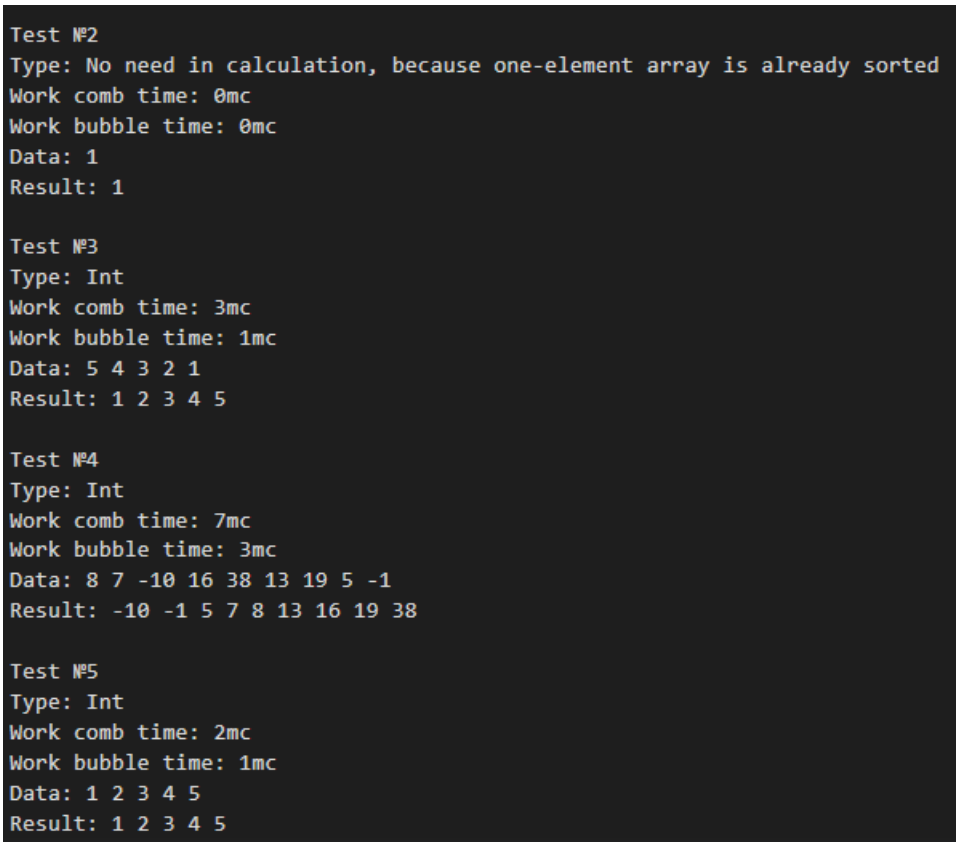

Рисунок 2 –Отладочные выводы программы

5. ТЕСТИРОВАНИЕ

INPUT	OUTPUT
2 7861879 917490 1746 816 -10 73971287 871794 7168 01 61710 461 11 1 1 1 1 2 2 -5 -5 -5 379 382 710 7394 9374 0284023 7392370 303710 -73 017 37037 366272 0 0 0 0 -1111111	-1111111 -73 -10 -5 -5 -5 0 0 0 0 1 1 1 1 1 2 2 2 11 17 379 382 461 710 816 1746 7168 7394 9374 37037 61710 284023 303710 366272 871794 917490 7392370 7861879 73971287
1	1
5 4 3 2 1	1 2 3 4 5
8 7 -10 16 38 13 19 5 -1	-10 -1 5 7 8 13 16 19 38
1 2 3 4 5	1 2 3 4 5
0 0 0 0 0 0	0 0 0 0 0 0
8.1 8.01 8.02 9.03 8.001 9.02 9.00001	8.001 8.01 8.02 8.1 9.00001 9.02 9.03
17 27 0 7 40	0 7 17 27 40
A 17 H AKDL	Array elements don't have same type
KKKK AAA A YH GHOSN	A AAA GHOSN KKKK YH
NDNG DKNbn JBG WNNJK GJNOANJ	DKNbn GJNOANJ JBG NDNG WNNJK
A B C D E	A B C D E
EEE EE E EEEE EEEEE	E EE EEE EEEE EEEEE
YO YOY YOYO YOYOY YOOOOOOOOOOOOO	YO YOOOOOOOOOOOOO YOY YOYO YOYOY
())(((())))) ((	((((()))) ()))(

ЗАКЛЮЧЕНИЕ

В ходе выполнения работы была написана программа, содержащая в себе реализацию сортировки расческой. Был получен опыт работы с дополнительными возможностями C++ и эффективной алгоритмизацией на нем. Также были закреплены знания полученные на протяжении семестра. Исходный код программы находится в приложении А.

ПРИЛОЖЕНИЕ А

СОДЕРЖИМОЕ ФАЙЛА COURSEWORK.CPP

```
#include "Source.h"

template <typename T1, typename T2>
void writeOutputData(std::ofstream& out, size_t testNumber, std::string type, T2
work_comb_time, T2 work_bubble_time, std::string const& arrStringForm,
                        std::vector<T1> const& arr)
{
      out << "Test №" << testNumber << "\n";
      out << "Type: " << type << "\n";
      out << "Work comb time: " << work_comb_time << "mc\n";
      out << "Work bubble time: " << work_bubble_time << "mc\n";
      out << "Data: " << arrStringForm << "\n";

      out << "Result: ";
      for (auto const& elem : arr)
            out << elem << " ";
      out << "\n\n";
}

void readInputData(std::ifstream& in, std::vector<std::string>& data)
{
      std::string tmp;
      while (std::getline(in, tmp))
      {
            if (tmp.back() == '\r')
                  tmp.erase(tmp.end() - 1);
            data.push_back(tmp);
      }
}

TypeCode determineType(std::string const& checkString)
{
      TransformPair<int> intTransform = from_string<int>(checkString);
      if (intTransform.transformResult == true)
            return TypeCode::TypeInt;

      TransformPair<char> charTransform = from_string<char>(checkString);
      if (charTransform.transformResult == true)
            return TypeCode::TypeChar;

      TransformPair<double> doubleTransform = from_string<double>(checkString);
      if (doubleTransform.transformResult == true)
            return TypeCode::TypeDouble;

      return TypeCode::TypeString;
}

int main(int argc, char** argv)
{
      if (argc > 2)
      {
            std::ifstream in(argv[1]);
            if (!in)
            {
                  std::cout << "Uncorrect input file";
                  return 0;
            }

            std::ofstream out(argv[2]);
```

```
	if (!out)
	{
		std::cout << "Uncorrect output file";
		return 0;
	}

	std::vector<std::string> fileData;
	readInputData(in, fileData);

	size_t testIndex = 1;
	for (auto const& line : fileData)
	{
		size_t firstSpaceInd = line.find(' ');
		if (firstSpaceInd == std::string::npos)
		{
			std::vector<std::string> arr = { line };
			writeOutputData(out, testIndex, "No need in calculation,
because one-element array is already sorted", 0, 0, line, arr);
			++testIndex;
			continue;
		}

		std::string firstElement(line.begin(), line.begin() +
firstSpaceInd);
		auto typeCalculationResult = determineType(firstElement);
		auto cmp = [](auto a, auto b) { return a < b; };

		switch (typeCalculationResult)
		{

		case TypeCode::TypeInt:
		{
			std::vector<int> arr;
			auto splitResult = split(arr, line, ' ');
			if (splitResult == false)
			{
				out << "Test №" << testIndex << "\n";
				out << "Uncorrect input data\n";
				++testIndex;
				continue;
			}

			auto start_comb = std::chrono::system_clock::now();
			comb(arr, cmp);
			auto end_comb = std::chrono::system_clock::now();

			auto start_bubble = std::chrono::system_clock::now();
			bubble(arr, cmp);
			auto end_bubble = std::chrono::system_clock::now();

			writeOutputData(out, testIndex, "Int",
std::chrono::duration_cast<std::chrono::microseconds>(end_comb -
start_comb).count(),

std::chrono::duration_cast<std::chrono::microseconds>(end_bubble -
start_bubble).count(), line, arr);

			break;
		}
		case TypeCode::TypeChar:
		{
			std::vector<char> arr;
			auto splitResult = split(arr, line, ' ');
```

```
                        if (splitResult == false)
                        {
                              out << "Test №" << testIndex << "\n";
                              out << "Uncorrect input data\n";
                              ++testIndex;
                              continue;
                        }

                        auto start_comb = std::chrono::system_clock::now();
                        comb(arr, cmp);
                        auto end_comb = std::chrono::system_clock::now();

                        auto start_bubble = std::chrono::system_clock::now();
                        bubble(arr, cmp);
                        auto end_bubble = std::chrono::system_clock::now();

                        writeOutputData(out, testIndex, "Char",
std::chrono::duration_cast<std::chrono::microseconds>(end_comb -
start_comb).count(),

      std::chrono::duration_cast<std::chrono::microseconds>(end_bubble -
start_bubble).count(), line, arr);

                        break;
                  }
                  case TypeCode::TypeDouble:
                  {
                        std::vector<double> arr;
                        auto splitResult = split(arr, line, ' ');
                        if (splitResult == false)
                        {
                              out << "Test №" << testIndex << "\n";
                              out << "Uncorrect input data\n";
                              ++testIndex;
                              continue;
                        }

                        auto start_comb = std::chrono::system_clock::now();
                        comb(arr, cmp);
                        auto end_comb = std::chrono::system_clock::now();

                        auto start_bubble = std::chrono::system_clock::now();
                        bubble(arr, cmp);
                        auto end_bubble = std::chrono::system_clock::now();

                        writeOutputData(out, testIndex, "Double",
std::chrono::duration_cast<std::chrono::microseconds>(end_comb -
start_comb).count(),

      std::chrono::duration_cast<std::chrono::microseconds>(end_bubble -
start_bubble).count(), line, arr);

                        break;
                  }
                  case TypeCode::TypeString:
                  {
                        std::vector<std::string> arr;
                        auto splitResult = split(arr, line, ' ');
                        if (splitResult == false)
                        {
                              out << "Test №" << testIndex << "\n";
                              out << "Uncorrect input data\n";
                              ++testIndex;
```

```
                              continue;
                        }

                        auto start_comb = std::chrono::system_clock::now();
                        comb(arr, cmp);
                        auto end_comb = std::chrono::system_clock::now();

                        auto start_bubble = std::chrono::system_clock::now();
                        bubble(arr, cmp);
                        auto end_bubble = std::chrono::system_clock::now();

                        writeOutputData(out, testIndex, "Double",
std::chrono::duration_cast<std::chrono::microseconds>(end_comb -
start_comb).count(),

      std::chrono::duration_cast<std::chrono::microseconds>(end_bubble -
start_bubble).count(), line, arr);

                        break;
                  }
                  default:
                        out << "Incorrect type!\n";
                        continue;
                  }
                  ++testIndex;
            }
      }

      return 0;
}
```

СОДЕРЖИМОЕ ФАЙЛА HEADER.H

```
#pragma once
#include <iostream>
#include <fstream>
#include <vector>
#include <string>
#include <chrono>
#include <sstream>

enum class TypeCode
{
      TypeInt,
      TypeChar,
      TypeDouble,
      TypeString
};

template<typename T>
struct TransformPair
{
      TransformPair(T newVal, bool newTranformResult) : value(newVal),
transformResult(newTranformResult)
      { }

      T value;
      bool transformResult;
};

template <typename T>
TransformPair<T> from_string(std::string const& checkString)
{
```

```
	T value;
	std::istringstream stream(checkString);
	stream >> value;

	if (stream.fail() || stream.peek() != EOF)
		return TransformPair<T>(value, false);

	return TransformPair<T>(value, true);
}

template <typename T>
bool split(std::vector<T>& arr, std::string const& arrStringForm, char
separator)
{
	auto startPosition = arrStringForm.begin();
	while (1)
	{
		auto searchResult = std::find(startPosition, arrStringForm.end(),
separator);
		if (searchResult == arrStringForm.end())
		{
			TransformPair<T> transformToT =
from_string<T>(std::string(startPosition, arrStringForm.end()));
			if (transformToT.transformResult == false)
				return false;

			arr.push_back(transformToT.value);
			return true;
		}
		TransformPair<T> transformToT =
from_string<T>(std::string(startPosition, searchResult));
		if (transformToT.transformResult == false)
			return false;

		arr.push_back(transformToT.value);

		startPosition = searchResult + 1;
	}
}


template <typename T, typename F>
void comb(std::vector<T>& arr, F cmp)
{
	int n = 0; // количество перестановок
	double fakt = 1.2473309; // фактор уменьшения
	double step = arr.size() - 1;

	while (step >= 1)
	{
		for (size_t i = 0; i + step < arr.size(); ++i)
		{
			if (cmp(arr[i + static_cast<size_t>(step)], arr[i]))
			{
				std::swap(arr[i + static_cast<size_t>(step)], arr[i]);
				n++;
			}
		}
		step /= fakt;
	}

	for (size_t i = 0; i < arr.size() - 1; i++)
	{
```

```
            bool swapped = false;
            for (size_t j = 0; j < arr.size() - i - 1; j++)
            {
                  if (cmp(arr[j + 1], arr[j])) {
                        std::swap(arr[j], arr[j + 1]);
                        swapped = true;
                        ++n;
                  }
            }

            if (!swapped)
                  break;
      }
}

template <typename T, typename F>
void bubble(std::vector<T>& arr, F cmp)
{
      for (size_t i = 0; i < arr.size() - 1; i++)
      {
            for (size_t j = 0; j < arr.size() - i - 1; j++)
            {
                  if (cmp(arr[j + 1], arr[j])) {
                        std::swap(arr[j], arr[j + 1]);
                  }
            }
      }
}
```